Des explications pour reconnaître et exploiter les structures cachées

Hadrien Cambazard **Narendra Jussien**

École des Mines de Nantes – LINA CNRS FRE 2729
4 rue Alfred Kastler – BP 20722 – F-44307 Nantes Cedex 3, France
hcambaza@emn.fr jussien@emn.fr

Résumé

L'identification de structures propres à un problème est souvent une étape clef pour la conception d'heuristiques de recherche comme la compréhension de la complexité du problème. De nombreuses approches en Recherche Opérationnelle emploient des stratégies de relaxations ou décompositions dès lors que certaines structures idoines ont été identifiées. L'étape suivante est la conception d'algorithmes de résolution qui puisse intégrer à la volée, pendant la résolution, ce type d'information. Cet article propose d'utiliser un solveur de contraintes à base d'explications pour collecter de l'information pertinente sur les structures dynamiques et statiques inhérentes au problème. Par ailleurs, la reconnaissance de relations spécifiques entre les variables suggère l'adaptation d'algorithmes dédiés issus du monde de la Recherche Opérationnelle au contexte de la programmation par contraintes. Une telle adaptation est discutée dans le cadre de la décomposition de Benders.

Abstract

Recent work have exhibited specific structure among combinatorial problem instances that could be used to speed up search or to help users understand the dynamic and static intimate structure of the problem being solved. Several Operations Research approaches apply decomposition or relaxation strategies upon such a structure identified within a given problem. The next step is to design algorithms that adaptatively integrate that kind of information during search. We claim in this paper that using an explanation-based constraint solver may lead to collect invaluable information on the intimate dynamic and static structure of a problem instance. We define several impact graphs to be used to design generic search guiding techniques and to identify hidden structures of instances. Finally, we discuss how dedicated OR solving strategies (such as Benders decomposition) could be adapted to constraint programming when specific relationships between variables are exhibited.

1 Introduction

Les stratégies de recherches génériques pour résoudre des problèmes d'optimisation combinatoires semblent le Saint-Graal pour les communautés RO et CP. Différentes pistes sont explorées à l'heure actuelle : adapter dynamiquement la manière dont le solveur s'emploie à résoudre un problème, identifier des structures spécifiques à une instance particulière pour accélérer la recherche, etc. Dans tous les cas, la clef est de pouvoir identifier, comprendre et utiliser les structures intimes présentes dans les instances d'un problème combinatoire donné [8, 18, 19]. Par le passé, Bessiere [6] a déjà cherché à prendre en compte le voisinage des variables. Refalo [17] a récemment introduit des stratégies d'impact pour la programmation par contraintes qui utilisent dynamiquement la structure du problème. Dans cet article, nous nous intéressons aux relations entretenues par les variables du problème. L'objectif est d'identifier et de différencier à la fois les structures dynamiques (créées par l'algorithme de recherche) et statiques (relatives à une instance) du problème. Une structure est entendue comme un sous-ensemble de variables qui joue un rôle spécifique au sein du problème. Nous définissons à cette fin, plusieurs mesures d'impacts ainsi que leur graphe correspondant de manière à pouvoir :

- reconnaître les structures cachées (invisibles dans le réseau de contraintes) ;
- concevoir des stratégies de recherche génériques ;
- examiner l'utilisation potentielle de l'analyse des impacts pour des méthodes procédant par décomposition, telle que la décomposition de Benders ;

Nos nouvelles mesures d'impact s'appuient sur le calcul d'explications qui fournit une information sur l'enchaînement des inférences effectuées au cours de

la propagation et donc implicitement des relations au sein de l'ensemble des variables.
L'article est organisé de la manière suivante : la section 2 introduit les bases et les motivations de notre travail ; différentes mesures d'impacts et leurs graphes associés sont présentés section 3 en distinguant leurs capacités respectives à refléter des structures dynamiques et statiques sur un exemple concret. Finalement, nous essayons de montrer leur intérêt à la fois comme heuristique de recherche par défaut mais aussi pour la conception d'algorithmes de résolution dédiés inspirés de la décomposition logique de Benders.

2 Les stratégies de recherche pour des problèmes structurés

Les stratégies de recherche efficaces exploitent certains aspects ou caractéristiques spécifiques d'un problème (ou d'une instance) donnée. En Recherche Opérationnelle, les stratégies de relaxations ou décompositions exploitent le fait qu'un problème sous-jacent (ou une partie du problème originel) peut être traité comme un problème classique (flot compatible ou maximum, plus courts chemins, sac à dos, etc.). Cet aspect du problème est souvent désigné sous le terme de *structure*.
Un problème est souvent dit structuré si ces composantes (variables et/ou contraintes) ne jouent pas toutes le même rôle, ou ne revêtent pas la même importance au sein du problème. La complexité d'un tel problème réside souvent dans l'interaction et l'influence mutuelle de ses différentes composantes. Une difficulté centrale dans l'identification de telles structures tient au fait qu'une structure n'existe pas nécessairement d'un point de vue statique (au niveau de l'instance). C'est souvent l'interaction entre une instance particulière et les choix de l'algorithme de recherche lui-même qui engendre une telle structure. Elle est désignée par la suite sous le nom de *structure dynamique* et correspond par exemple à de mauvais choix initiaux ou à la création de nouvelles relations entre les variables dues à l'ajout de nouvelles contraintes par la recherche.
Les *backdoors*, récemment introduits en [19] sont un concept intéressant pour caractériser des structures cachées au sein d'un problème. Ils sont définis d'un point de vue informel comme des sous-ensembles de variables qui encapsulent toute la combinatoire du problème : une fois cet ensemble instancié, le sous-problème restant peut être résolu efficacement. Différentes stratégies de recherche s'appuient sur ce principe. Les deux suivantes sont au cœur de notre démarche :

- les heuristiques de branchement en programmation par contraintes s'efforcent d'orienter la recherche le plus tôt possible vers des variables constituant un *backdoors* dans la mesure où les choix ont pour but de simplifier au maximum le problème. Elles sont basées sur une idée simple : sélectionner une variable qui contraint au maximum le problème et mène sur un espace de recherche aussi petit que possible. Ce principe (couramment désigné par le terme de *first-fail*) est souvent mis en œuvre en prenant en compte le degré et le domaine courant des variables (voir [3],[6] pour des variantes plus pertinentes). Plus récemment, Refalo [17], a proposé de caractériser l'impact d'un choix et d'une variable en examinant en moyenne la réduction de l'espace de recherche engendrée par ce choix (une autre manière d'identifier des *backdoors*).
- la décomposition de Benders [2] s'apparente également à une stratégie s'appuyant sur l'existence de *backdoors*. C'est une stratégie de résolution basée sur une partition du problème selon ses variables en deux ensembles x et y. Un problème maître fournit une affectation réalisable x^* et un sous-problème essaye de compléter cette affectation sur les variables y. Le sous-problème produit alors une coupe (une contrainte) ajoutée au problème maître de façon à éliminer l'affectation x^* courante. Les coupes intéressantes sont celles qui permettent d'éliminer non seulement x^* mais également toutes les classes d'affectation infaisables ou sous-optimales pour les même raisons que x^*. Cette technique est dédiée à des problèmes qui présentent des structures spécifiques. Le problème maître est basé sur un ensemble pertinent de variables qui vérifie en général les hypothèses suivantes :
  1. le sous-problème résultant est simple. En pratique, plusieurs sous-problèmes de petites tailles sont utilisés, simplifiant la recherche souvent exhaustive nécessaire à la production de la coupe.
  2. la coupe de Benders est suffisamment pertinente pour garantir une convergence rapide de l'ensemble de la technique.

  Dans une telle décomposition, les variables du problème maître constituent un *backdoors*. En effet, l'hypothèse 1 nous assure que le sous-problème peut être résolu efficacement une fois le maître instancié. Par ailleurs, si le sous-problème résultant peut être résolu de manière polynomiale (il est dans ce cas analogue à un *strong backdoors*), une coupe plus puissante correspondant au conflit minimal peut souvent être extraite.

Pour la dernière technique abordée, les structures demandent à être identifiées avant que la recherche ne commence. D'ordinaire, c'est l'analyse du réseau

de contraintes qui permet d'extraire certaines caractéristiques intéressantes d'une instance. Ainsi, il est classique en coloration de graphes de commencer par rechercher les cliques de taille maximum pour fournir à la fois des bornes inférieures et éventuellement renforcer la propagation en ajoutant des contraintes globales du type *all-different*. Une telle analyse ne procure qu'une information sur les structures statiques. Néanmoins, des structures statiques cachées ou dynamiques sont d'un grand intérêt pour de nombreuses stratégies de recherche. Naturellement leur identification est au moins aussi coûteuse que la résolution du problème d'origine. Nous pensons que la propagation fournit une information qui devrait permettre d'approximer de telles structures. Le calcul des explications est certainement une manière d'extraire cette information de la phase de propagation.

3 Des explications au cœur des structures

Refalo [17] a introduit une mesure d'impact dans l'objectif de détecter les choix qui engendrent la plus grande réduction de l'espace de recherche. Il propose donc de caractériser l'impact d'une décision en calculant le produit cartésien des domaines avant et après la décision en question. Nous chercherons à aller plus loin en analysant où cette réduction intervient et dans quelle mesure les choix antérieurs sont impliqués. Nous étendrons ces mesures à un graphe d'impact entre variables, prenant à la fois en compte l'effet des décisions passées comme la réelle implication d'un choix dans chaque inférence effectuée pendant la résolution.
Notre objectif est d'identifier des variables qui contraignent au maximum le problème, des sous-ensembles de variables particuliers qui entretiennent de fortes relations ou qui ont un impact important sur l'ensemble du problème (analogue à un *backdoors*). Nous avons axé notre étude sur les points suivants :

- l'impact ou l'influence d'une variable sur la réduction directe de l'espace de recherche ;
- l'impact d'une variable au sein d'une chaîne de déductions même longtemps après que cette variable ait été instanciée ;
- la région de l'espace de recherche sur laquelle s'exerce l'influence d'une variable ainsi que les relations précises entre variables ;

Une telle information est liée au concept d'explications pour la programmation par contraintes.

3.1 Explications et programmation par contraintes

Les explications ont été introduites initialement pour améliorer les algorithmes de recherche arborescente basés sur le backtrack. Elles ont été utilisées plus récemment pour d'autres objectifs comprenant notamment la résolution dynamique de problèmes et l'interaction avec l'utilisateur [13].

Definition 1 *Une explication enregistre toute l'information nécessaire pour justifier le comportement du solveur (une réduction de domaine ou une contradiction). Elle est constituée d'un ensemble de contraintes C' (sous-ensemble des contraintes originelles C) et d'un ensemble de décisions dc_1, dc_2, ... prises durant la recherche. L'explication du retrait de la valeur a de la variable v s'écrit ainsi :*

$$C' \wedge dc_1 \wedge dc_2 \wedge \cdots \wedge dc_n \Rightarrow v \neq a$$

Les explications calculées par le solveur représentent la chaîne logique des inférences consécutives faites par le solveur au cours de la propagation. D'une certaine manière, elles fournissent une trace du comportement du solveur car toutes les opérations se doivent d'être *expliquées*. Par la suite, on notera E_i^{val} l'ensemble des explications[1] calculées depuis le début de la recherche pour tous les retraits de la valeur val de la variable i.

3.2 Caractériser l'impact

L'impact d'une décision $x_i = a$ s'exprime en accord avec le principe du *first fail*, à travers la réduction moyenne de l'espace de recherche engendrée par cette décision [17]. Néanmoins, cette réduction n'intervient pas uniquement au moment où la décision est imposée au problème mais aussi quand d'autres déductions (futures) qui sont en partie basées sur l'hypothèse $x_i = a$ sont faites. L'utilisation d'explications fournit donc une information supplémentaire sur l'implication réelle d'une déduction dans un retrait. Une décision passée $x_i = a$ a un impact effectif (du point de vue du solveur) sur une valeur val d'une variable x_j si elle apparaît dans l'explication justifiant ce retrait. On note $I_\alpha(x_i = a, x_j, val)$, l'impact sur la valeur val de x_j de la décision $x_i = a$. α est l'index utilisé pour distinguer les différentes mesures.
Notre première approche consiste à considérer le nombre de fois qu'une décision apparaît dans les explications calculées pour la valeur val de x_j. La taille de l'explication entre également en jeu. En effet, plus une explication est petite, plus elle est précise car plus le nombre d'hypothèses nécessaires à la déduction est réduit. Des explications de petites tailles entraînent de fortes relations.

$$I_0(x_i = a, x_j, val) = \sum_{\{e \in E_j^{val}, x_i = a \in e\}} 1/\left|e\right|$$

[1] le maintien de cette information est effectué à la volée et n'implique pas de conserver en mémoire toutes ces explications.

Différentes mesures basées tout d'abord sur l'activité du solveur et le calcul des explications (mesures I_1 et I_2) sont introduites à partir de cette mesure initiale, avec l'objectif de rendre compte des structures dynamiques. Dans un second temps, une mesure s'appuyant plus sur la réduction de l'espace de recherche (I_3) est introduite pour capturer les structures statiques et aider à guider la recherche. Comme la recherche oriente à l'évidence la propagation (et vice-versa), il semble assez naturel de normaliser cette mesure par rapport à la recherche.

- L'impact est ici normalisé par rapport au nombre de fois où une décision $x_i = a$ est prise : $|x_i = a|$. Le but étant simplement de distinguer les décisions fréquentes (*i.e* les plus récentes) et celles qui sont rarement re-considérées (les plus anciennes) :

$$I_1(x_i = a, x_j, val) = \frac{\sum_{\{e \in E_j^{val}, x_i = a \in e\}} \frac{1}{|e|}}{|x_i = a|}$$

- Une autre manière de normaliser est de considérer l'âge a_e^d d'une décision d au moment du calcul de l'explication e avec l'objectif de faire décroître l'impact des vieilles décisions. On obtient :

$$I_2(x_i = a, x_j, val) = \sum_{\{e \in E_j^{val}, x_i = a \in e\}} \frac{1}{|e| \times a_e^{x_i = a}}$$

- Le calcul des impacts est dispersé au sein du processus de résolution à chaque calcul d'une explication. C'est une approche sensiblement différente à [17] qui analyse chaque décision séparément pour évaluer son impact instantané. I_3 essaie d'identifier les réductions récurrentes de l'espace de recherche liées à une décision :

$$I_3(x_i = a, x_j, val) = \frac{\sum_{e \in E_j^{val}, x_i = a \in e} \frac{1}{|e|}}{|\{x_i = a \text{ active } \wedge val \in Dom(x_j)\}|}$$

$I_3(x_i = a, x_j, val)$ peut être considérée comme la probabilité que la valeur val de x_j soit éliminée si la décision $x_i = a$ est prise. Une telle mesure est mise à jour chaque fois qu'un nouveau retrait intervient dès lors que $x_i = a$ est active. Elle prend en compte la fréquence comme la proportion avec lesquelles une décision est impliquée dans une explication de retrait.

Ces impacts restent fortement dépendants de l'exploration effectuée et des techniques semblables à celles présentées dans [17] seront employées afin de les initialiser et de les affiner.

3.3 Relations entretenues par les variables

On peut introduire différents graphes d'impact sous la forme de graphes pondérés $GI(V, E)$ avec pour poids sur les arcs $I(x, y)$ pour tout couple $(x, y) \in E = V \times V$. De manière à définir les poids, les mesures d'impact introduites précédemment sont agrégées sur l'ensemble des valeurs du domaine :

$$I(x_i = a, x_j) = \sum_{val \in D(x_j)} I(x_i = a, x_j, val)$$

$I(x_i = a, x_j, val)$ peut être remplacé ici par l'une des 4 mesures parmi $(\{I_\alpha \mid \alpha \in [0, 1, 2]\})$. Un cas particulier se présente pour I_3 où il s'agit de relier l'impact à la réduction de l'espace de recherche engendrée par une variable sur une autre. On considère dès lors la taille du domaine initial :

$$I(x_i = a, x_j) = \frac{|D(x_j)| - \sum_{val \in D(x_j)} (1 - I_3(x_i = a, x_j, val))}{|D(x_j)|}$$

Dans ce contexte, $1 - I_3(x_i = a, x_j, val)$ correspond en quelque sorte à la probabilité de présence de la valeur val de la variable x_j après avoir pris la décision $x_i = a$. Le poids d'un arc peut à présent être calculé :

$$I(x_i, x_j) = \sum_{v \in D(x_i)} I(x_i = v, x_j)$$

3.4 Utilisation pour le branchement

Pour les mesures I_0, I_1 et I_2, l'impact global d'une décision est calculée en accumulant l'impact sur les variables de l'ensemble du problème :

$$I(x_i = a) = \sum_{x_j \in V} I(x_i = a, x_j)$$

De manière similaire au 3.3, l'attention est portée pour I_3 sur l'espace potentiel restant après un choix. La taille de l'espace courant P est évaluée comme le produit cartésien des domaines courants. Ainsi, l'impact global d'une décision sur le problème s'exprime directement comme la réduction de l'espace de recherche en considérant l'espace restant probable après la décision (de manière similaire à l'impact de Refalo [17], $(P_{before} - P_{after})/P_{before}$) :

$$I(x_i = a) = (P - \prod_{x_j \in V} \sum_{val \in D(x_j)} (1 - I_3(x_i = a, x_j, val)))/P$$

3.5 Un exemple illustratif

Une instance particulière issue du benchmark ultérieur introduit en section 4 est exhibé ici de façon à illustrer quels types de structures sont isolés par les

différentes mesures d'impact. De plus, nous examinerons comment l'information visuelle extraite du graphe d'impact permet à l'utilisateur d'effectuer une analyse du problème ainsi que sa résolution.
Nous considérons un problème binaire aléatoire dans lequel une structure est insérée en augmentant la dureté de certaines contraintes de manière à faire apparaître plusieurs sous-ensembles de variables entretenant des relations fortes. Les instances aléatoires sont caractérisées par un tuple $< N, D, p1, p2 >$ (on utilise le modèle B [1]) où N est le nombre de variables, D la taille des domaines, p_1 la densité du réseau et p_2 la dureté des contraintes. Trois sous-ensembles de 10 variables sont insérés avec une dureté de $p_2 = 53\%$ tandis que le reste du réseau est fixé à 3%.
L'instance spécifique isolée ici à titre d'illustration nous a semblé intéressante à cause de sa difficulté inattendue pour `mindom` [9]. En utilisant les différents graphes d'impact introduits précédemment, nous souhaitons répondre à différentes questions soulevées face à ce problème :

- est-il possible de reconnaître la structure intégrée au problème ?
- pourquoi `mindom` échoue sur cette instance ? Cette difficulté est-elle intrinsèque au problème ou à l'heuristique elle-même ?

Visualiser le graphe d'impact Les figures 1 à 5 donnent une représentation du graphe d'impact GI des 30 variables du problème. Nous utilisons une représentation sous forme de matrice d'adjacence [7]. Les cellules à l'intersection de chaque ligne i et colonne j indiquent l'impact de v_j sur v_i. Plus l'impact est fort, plus l'arc possède un poids élevé et plus la cellule est foncée. La matrice est ordonnée dans l'ordre où sont créés les noyaux cachés de variables[2].
Avant de commencer la recherche, une phase de singleton consistance est appliquée (chaque valeur de chaque variable est propagée [16]) afin d'initialiser l'impact des variables de manière homogène. Bien que le graphe soit entièrement connecté, la visualisation sous forme de matrice donnée en figure 1 permet de distinguer clairement les trois noyaux de variables qui entretiennent des relations fortes, juste après cette première étape de propagation (la mesure I_0 est utilisée ici).

La figure 2 donne une image du graphe d'impact après 2 minutes de recherche en utilisant `mindom` comme choix de variable. On peut noter que I_0 est centrée sur les aspects dynamiques de la recherche (les clusters initiaux ne sont plus du tout visibles comparés à la figure 1) alors que I_3 est focalisée sur les structures statiques et relègue dans l'ombre les liens faibles même après 2 minutes de calculs (voir figure 3). La zone la plus sombre dans le coin en bas à gauche de de la figure 2, montre que, selon I_0, les variables des deux premiers noyaux ont apparemment une forte influence sur le troisième noyau. On peut penser que de mauvais choix initiaux sur les deux premiers ensembles ont mené le solveur dans une succession d'échecs sur le troisième ensemble masquant ainsi les structures inhérentes au problème.

[2]Nous nous penchons actuellement sur la l'utilisation d'algorithmes de clustering de manière à identifier cet ordre particulier à partir du graphe d'impact seul.

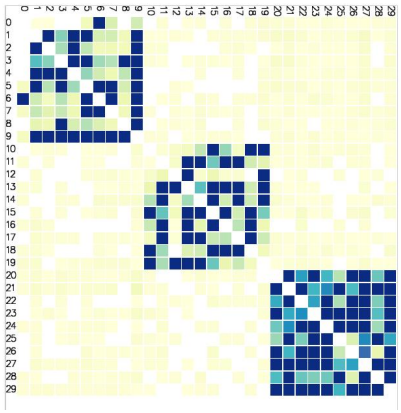

FIG. 1 – Une représentation du graphe d'impact à l'issue de la phase d'initialisation en utilisant la mesure I_0 comme mesure d'impact.

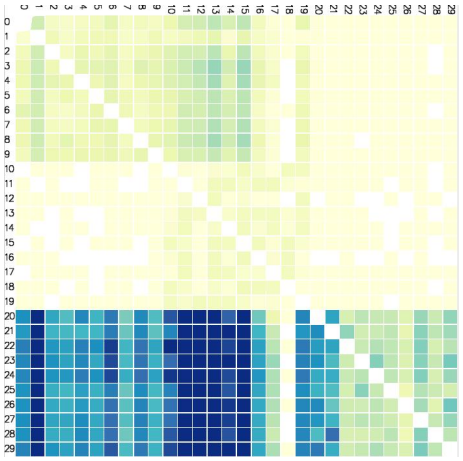

FIG. 2 – Le graphe d'impact basé sur I_0 après deux minutes de calcul en utilisant `mindom`

La figure 4 représente un graphe normalisé où l'influence d'une décision prise par le solveur est divisée par le nombre de fois où cette décision est prise pendant la résolution (mesure I_1). L'objectif est de pousser plus loin l'analyse précédente en distinguant deux types de décisions parmi celles qui possèdent une forte influence : certaines sont répétées fréquem-

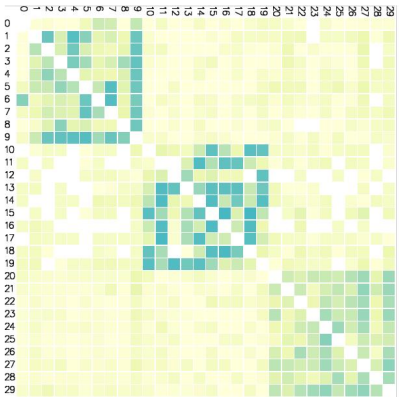

FIG. 3 – Le graphe d'impact basé sur I_3 après deux minutes de calcul en utilisant `mindom`

ment alors que d'autres ont guidé le solveur dans une branche stérile de l'espace de recherche et apparaissent dès lors dans de nombreuses explications d'échecs. On cherche à isoler de cette manière les mauvais choix qui semblent se situer sur le deuxième noyau.

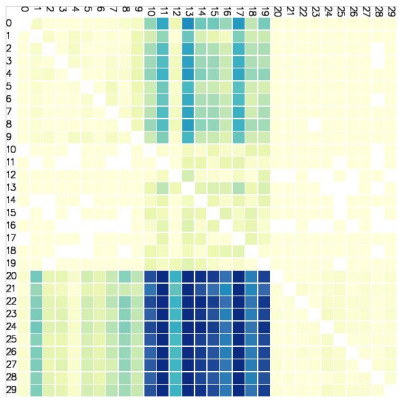

FIG. 4 – Une représentation du graphe d'impact normalisé par rapport au nombre de fois où une décision est prise (I_1).

Enfin la figure 5 rapporte l'activité de la résolution en s'appuyant sur un graphe d'impact dans lequel l'effet des anciennes décisions est progressivement éliminé (mesure I_2). Il apparaît alors clairement que le solveur ne cesse d'échouer sur le premier et le troisième noyau avec très peu d'interaction sur le deuxième où se situent les mauvais choix initiaux. De manière à confirmer cette interprétation, nous avons adapté notre heuristique de recherche pour prendre en compte l'impact des variables au cours de la résolution et revenir ainsi immédiatement sur les variables dont l'influence s'accroît de manière inconsidérée (parce qu'elles interviennent dans toutes les explications de retrait sans fournir une réduction significative de l'espace de recherche). Le problème est alors résolu instantanément.

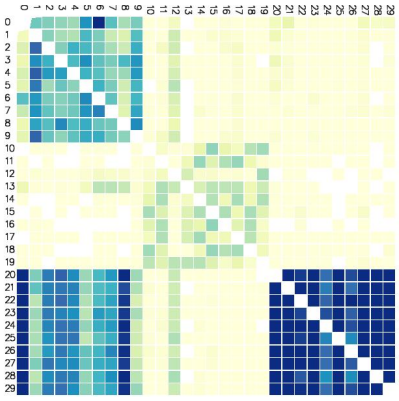

FIG. 5 – Une représentation du graphe d'impact normalisé sur l'âge des décisions (I_2)

4 Les impacts comme heuristiques de recherche

Nous cherchons dans cette section à affiner notre compréhension de la recherche et à illustrer comment l'impact introduit précédemment constitue une piste sérieuse pour améliorer les techniques de recherche génériques. La variable x choisie pour le branchement est celle qui maximise $\sum_{a \in D(x)} I(x = a)$, la valeur choisie est celle qui minimise $I(x = a)$ de manière à favoriser les combinaisons futures consistantes ($D(x)$ correspond ici au domaine courant de x). Les égalités sont arbitrées aléatoirement. Les impacts sont par ailleurs initialisés à l'aide d'une phase de propagation analogue à de la singleton-consistance. Nos expérimentations sont conduites sur un Pentium 4, 3 GigaHz, sous Windows XP. Notre solveur de contraintes est la version java la plus récente de `choco` (`choco.sf.net`). La recherche n'est pas limitée à une recherche arborescente classique, les explications étant également exploitées pour remonter directement à l'origine des conflits (à la `mac-cbj`). En pratique, le comportement est très proche de `mac` avec le coût supplémentaire du maintien des explications. Nous avons considéré trois types de benchmark :

1. Le premier jeu de tests provient des expérimentations de Refalo [17] : un ensemble de problèmes de multiknapsack modélisés par des variables binaires. Une limite de temps est fixée à 1500s. Nous nous concentrons ici sur le nombre de nœuds[3] de la recherche qui est directement relié à la pertinence de la mesure. De plus, comme les égalités sont arbitrées aléatoirement, une moyenne sur une

[3]Dans les résultats présentés, quand une technique de restart est utilisée, seul le nombre de nœuds de la dernière exécution est indiqué contrairement au temps qui correspond au temps global.

dizaine d'exécutions est reportée.

2. Le deuxième jeu de tests est constitué par un problème binaire aléatoire du modèle B classique (voir section 3.5) avec les paramètres suivants $< 50, 10, 30, p_2 >$. On considère le nombre d'instances résolues dans une limite de temps de 120s pour 100 problèmes à chaque valeur de p_2.
3. Le dernier ensemble de tests est fait de problèmes binaires aléatoires structurés décrit à la section 3.5. Un problème $< 45, 10, 35, p_2 >$ est structuré avec trois noyaux de 15 variables reliés par une dureté intra-noyaux p_2 et une dureté inter-noyaux de 3%.

Trois mesures ($\{I_\alpha \mid \alpha \in [1, 2, 3]\}$) sont comparées ainsi que notre implémentation de la mesure introduite dans [17] (dénotée I_{ref}). Comme la mesure spécifie intégralement la recherche effectuée, nous nous référerons par la suite aux stratégies I_α et I_{ref}.

4.1 Premier benchmark : problèmes de multiknapsack

Sur ce premier benchmark (dont les résultats sont indiqués table 1), I_{ref} apparaît comme la meilleure stratégie. L'utilisation d'explications semble apporter une information pertinente mais au terme d'un long apprentissage (elle demande des *restarts*) et s'avère beaucoup plus coûteuse en temps sur ce problème où près d'un million de nœuds doivent être explorés. Le nombre de nœuds de `mindom` est indiqué ici à titre de référence.

I_1 et I_2 ne sont pas pertinentes sur ces instances et exigent peut-être une politique de restart très fine dans la mesure où elles essayent de détecter de mauvais points de choix et sont centrées sur la dynamique de la recherche. Comme mentionné par Refalo, l'utilisation de restarts augmente uniquement le temps global d'exécution de I_{ref} mais semble important pour I_3. I_3 se présente en effet comme une mesure plus fine qui demande certainement plus de temps avant de se révéler pertinente pour la recherche.

4.2 Second benchmark : problèmes aléatoires binaires

Sur ce benchmark volontairement non structuré où la taille des domaines (variables entières et non binaires) offre à `mindom` de meilleures chances d'évaluer les variables les plus contraintes. Les résultats (visibles sur la figure 6 ainsi qu'en pourcentage d'instances résolues sur le tableau 2) ne sont pas du tout en faveur des impacts seuls. En revanche, leur combinaison avec `mindom` comme moyen d'arbitrer les égalités est très fructueuse et améliore considérablement les résultats en permettant de résoudre près de 92.9% des instances sur l'ensemble de la transition de phase contre 79% pour `mindom`. Cette combinaison limite les mauvais choix initiaux et fait de I_{ref} la meilleure technique alors qu'elle se plaçait derrière I_2 et I_1 (qui résout près de 17% d'instances supplémentaires) dans le cas contraire. L'utilisation de restart semble ici inutile et augmente généralement le temps global d'exécution.

TAB. 2 – Impacts sur les problèmes binaires aléatoires

`stratégie`	% d'instances résolues
I_{ref}	36 %
I_1	53.7 %
mindom	79 %
mindom + I_2	86 %
mindom + I_{ref}	92.9 %

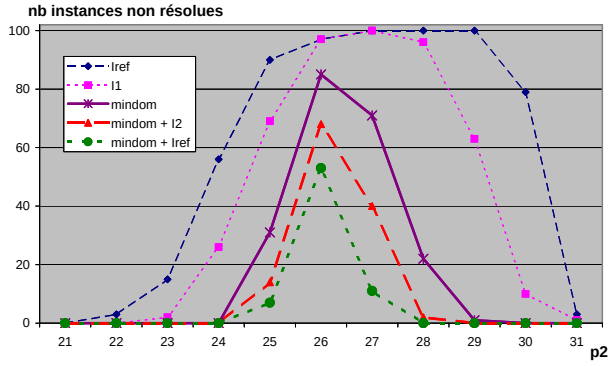

FIG. 6 – Nombre d'instances non résolues pour une sélection de mesures d'impact sur des `CSP` binaires aléatoires.

4.3 Troisième benchmark : problèmes binaires aléatoires structurés

Sur cet exemple, la stratégie I_{ref} semble rencontrer des difficultés même comparée à `mindom`. La figure 7 reporte le nombre d'instances non résolues dans la limite de temps de 120 secondes. À nouveau, le *restart* ne favorise pas I_{ref} mais s'avère effectif pour I_3. Les meilleurs résultats uniquement (*i.e* obtenus avec ou sans *restart*) sont donc indiqués pour chaque technique. Les résultats les plus impressionnants sont obtenus cette fois en se focalisant sur les aspects dynamiques de la structure des instances (*i.e* en utilisant les stratégies I_1 et I_2). Les instances sont intégralement résolues de cette manière. On peut également noter que I_3 donne des résultats supérieurs à I_{ref} malgré son coût important.

TAB. 1 – Impacts sur les problèmes de multiknapsack

	mindom	I_{ref}		I_3		I_3+rest	
	Noeuds	Temps	Noeuds	Temps	Noeuds	Temps	Noeuds
mknap1-2	38	0	25.9	0	23.1	0	23.1
mknap1-3	385	0.1	188.7	0.3	354.1	0.3	255.2
mknap1-4	16947	0.7	982.7	4.2	2754	3.2	979.5
mknap1-5	99003	11.2	21439.6	229.1	110666.4	112.8	20237.4
mknap1-6	21532776	425.7	612068	> 1500		> 1500	

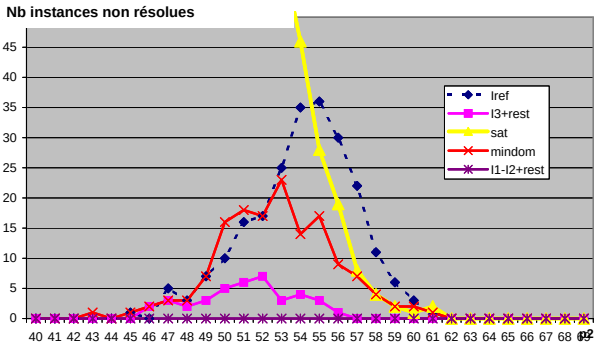

FIG. 7 – Nombre d'instances non résolues pour différentes mesures d'impact sur des CSP binaires aléatoires avec le nombre d'instances satisfiables (sat).

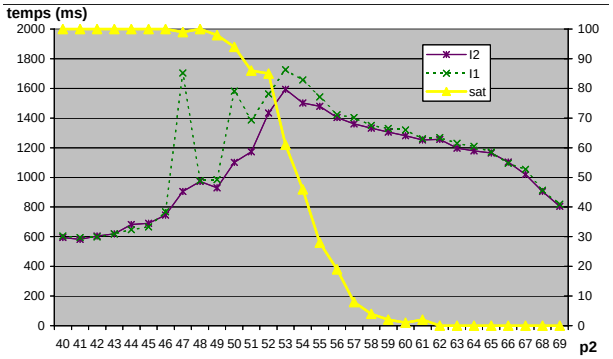

FIG. 8 – Temps de résolution pour I_1 et I_2 sur des CSP binaires aléatoires avec le nombre d'instances faisables (sat).

Le succès d'I_1 et I_2 est peut-être du au fait que la complexité de ce benchmark ne réside pas uniquement au niveau des instances mais aussi certainement au degré élevé d'interaction avec l'algorithme de recherche. La présence de telles structures artificielles favorise de notre point de vue un certain type de comportement *heavy-tailed* [18] qui rend les premiers choix critiques. Il peut en effet être noté sur la figure 8 que I_1 et I_2 sont sujets à de mauvais comportements qui peuvent apparaître en dehors de la transition de phase. Le même phénomène sur une plus large échelle peut être à l'origine des mauvais résultats de I_{ref}.

4.4 Premières investigations sur les stratégies à base d'impacts

I_1 et I_2 sont fortement basées sur l'activité du solveur pendant la recherche (en se focalisant donc sur la partie dynamique des structures). Leur utilisation peut s'avérer extrêmement rentable dans la mesure où elles révèlent les mauvais choix initiaux (dont l'influence grandit démesurément au cours de la recherche sans apporter un élagage utile puisque le solveur ne parvient pas à revenir dessus).
I_3 est trop coûteuse (en terme de temps) dans l'état actuel pour être utilisée comme heuristique par défaut mais des compromis intéressants entre I_{ref} et I_3 peuvent être imaginés de manière à profiter de la robustesse générale de I_{ref} tout en essayant de se prémunir contre des comportements *heavy-tailed* liés au choix initiaux. Malgré le côté très artificiel de ces expérimentations, elles cherchent à affiner notre compréhension des phénomènes mis en jeu pendant la recherche et mettent en évidence l'intérêt pour I_{ref} de s'appuyer aussi sur une information plus fine provenant du calcul des explications.

5 Exploitation de structures spécifiques

La décomposition de Benders en Recherche Opérationnelle est une technique dédiée à des problèmes qui possèdent une structure statique présentant un sous ensemble de variables avec un impact important sur l'ensemble du problème et où, idéalement, le reste du problème se décompose en sous-problèmes indépendants. La décomposition utilise une relation maître-esclave entre variables et les variables du maître sont appelées *variables compliquantes* dès 1972 par Goeffrion [20] de sorte qu'une fois ces variables instanciées, le sous-problème d'optimisation résultant est beaucoup plus simple. On souhaite mettre en œuvre ce type de décomposition qui a largement démontré son efficacité en Recherche Opérationnelle dans le contexte

de structures *cachées* et révélées par le graphe d'impact (sous la forme de sous ensembles ayant de fortes intra-relations et de faibles inter-relations). La mise en œuvre d'une telle technique en programmation par Contraintes n'est néanmoins pas immédiate. Traditionnellement, les coupes de Benders sont limitées à la programmation linéaire et calculée par la résolution du dual du sous-problème[4]. Elles exigent donc que les variables duales puissent être définies pour appliquer la décomposition. Cependant, Hooker et Ottosson [10] propose de dépasser cette limite et d'élargir le concept de dualité en introduisant un *dual logique* valable pour tout type de sous-problème. Ils se réfèrent à un schéma plus général et suggèrent une manière différente de penser la dualité en s'appuyant sur la logique, la capacité de produire une preuve et un ensemble d'hypothèses suffisantes pour cette preuve.
Néanmoins, ce dual logique demande à être implémenté pour chaque classe de sous-problèmes en identifiant les coupes proprement dites [11, 4]. On peut considérer le dual comme un certificat d'optimalité, autrement dit, une explication (comme introduit au 3.1) d'inconsistance dans notre cas. Le paradigme de la programmation par contraintes expliquée fournit d'une certaine manière une implémentation générique de la décomposition de Benders généralisée dans le cas de problèmes de satisfaction [4]. On peut noter ici que le calcul des explications est paresseux[5], la première explication trouvée est conservée alors qu'elle n'est pas unique. Le calcul de l'explication minimale à la volée pendant la résolution est évidemment trop coûteux. Par conséquent, un tel dual logique fournit une solution duale arbitraire[6] mais pas nécessairement la solution optimale. À l'évidence, le succès d'une telle approche dépend du degré avec lequel des explications précises peuvent être calculées pour les contraintes du sous-problème.
La programmation par contraintes à base d'explications propose des algorithmes du type de `mac-dbt` [14] ou `decision-repair` [15] qui essayent de se focaliser naturellement sur le problème maître d'une telle décomposition mais peuvent revenir à un comportement plus conventionnel. L'étape suivante consiste à utiliser la structure exhibée par le graphe d'impacts pour appliquer un schéma de décomposition dans une seconde phase de résolution. L'identification de sous-problèmes pourrait par ailleurs guider la génération de coupes pour le problème maître de manière à rassembler autant d'information que possible là où réside la vraie combinatoire du problème.

[4] En référence à la dualité dans un contexte linéaire.

[5] L'ensemble des explications n'est pas calculé suite à un retrait de valeur. Seule, celle correspondant au raisonnement courant du solveur est conservée

[6] Ce qui est également vrai en programmation linéaire où toute solution duale constitue une borne pour le primal.

6 Conclusion

Nous avons introduit dans cet article plusieurs indicateurs qui peuvent s'avérer utile à la fois pour l'identification et l'utilisation de certaines structures clef au cœur d'un problème combinatoire. Nous nous sommes intéressés dans cette étude aux relations entre les variables en ouvrant de nouvelles perspectives sur la conception d'heuristiques de recherche générique pour la programmation par contraintes comme la conception de nouveaux algorithmes qui pourraient être proposés par les solveurs. Nous pensons que la présence de *backdoors* ou de sous-ensembles de variables présentant un impact important sur l'ensemble du problème pourrait être utilisé explicitement par des stratégies de décomposition *ad hoc* inspirées de la Recherche Opérationnelle. La décomposition de Benders et son extension logique en est un exemple. Il s'agit en effet d'une technique de *backdoors* qui pourrait être appliquée en programmation par contraintes comme une stratégie d'enregistrement de *nogood*.

Références

[1] D. Achlioptas, L. Kirousis, E. Kranakis, D. Krizanc, M. Molloy, and Y. Stamatiou. Random constraint satisfaction : a more accurate picture. In *Proceedings CP 1997*, pages 121–135, Linz, Austria, 1997.

[2] J. F. Benders. Partitionning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4 :238–252, 1962.

[3] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proceedings ECAI'04*, pages 482–486, 2004.

[4] Hadrien Cambazard, Pierre-Emmanuel Hladik, Anne-Marie Déplanche, Narendra Jussien, and Yvon Trinquet. Decomposition and learning for a real time task allocation problem. In *Proceedings CP 2004*, pages 153–167, 2004.

[5] G. Cleuziou, L. Martin, and C. Vrain. Disjunctive learning with a soft-clustering method. In *ILP'03 :13th International Conference on Inductive Logic Programming*, pages 75–92. LNCS, September 2003.

[6] C. Bessière, A. Chmeiss and L. Saïs. Neighborhood-based variable ordering heuristics for the constraint satisfaction problem In *Proceeding CP'01*, pages 565–569, Paphos, Cyprus, 2001, Short paper.

[7] Mohammad Ghoniem, Narendra Jussien, and Jean-Daniel Fekete. VISEXP : visualizing constraint solver dynamics using explanations. In *Proceedings FLAIRS'04*, Miami, Florida, USA, May 2004.

[8] Carla P. Gomes, Bart Selman, and Nuno Crato. Heavy-tailed distributions in combinatorial search. In *Proceeding CP 97*, pages 121–135, Linz, Austria, 1997.

[9] R. Haralick and G. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial intelligence*, 14(9) :263–313, 1980.

[10] J.N. Hooker and G. Ottosson. Logic-based benders decomposition. *Mathematical Programming*, 96 :33–60, 2003.

[11] Vipul Jain and I. E. Grossmann. Algorithms for hybrid MILP/CP models for a class of optimization problems. *INFORMS Journal on Computing*, 13 :258–276, 2001.

[12] Narendra Jussien. *The versatility of using explanations within constraint programming.* Habilitation thesis, Université de Nantes, France, 2003. also available as RR-03-04 research report at École des Mines de Nantes.

[13] Narendra Jussien and Vincent Barichard. The PaLM system : explanation-based constraint programming. In *Proceedings of TRICS : Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000*, pages 118–133, Singapore, September 2000.

[14] Narendra Jussien, Romuald Debruyne, and Patrice Boizumault. Maintaining arc-consistency within dynamic backtracking. In *Proceedings CP 2000*, pages 249–261, Singapore, 2000. Springer-Verlag.

[15] Narendra Jussien and Olivier Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139(1) :21–45, July 2002.

[16] P. Prosser, K. Stergiou, and T. Walsh. Singleton consistencies. In R. Dechter, editor, *Proceedings CP 2000*, pages 353–368, Singapore, 2000.

[17] Philippe Refalo. Impact-based search strategies for constraint programming. In *Proceedings CP 2004*, pages 556–571, Toronto, Canada, 2004.

[18] Ryan Williams, Carla Gomes, and Bart Selman. On the connections between backdoors and heavy-tails on combinatorial search. In *the International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2003.

[19] Ryan Williams, Carla P. Gomes, and Bart Selman. Backdoors to typical case complexity. In *Proceedings IJCAI 2003*, 2003.

[20] A. M. Geoffrion. Generalized Benders Decomposition. *Journal of Optimization Theory And Practice*, Vol. 10, No. 4, 1972.